String
We will continue with the functionalities of String

Return Type	**Method**	**Function**
char	charAt(`int index`)	The char at a location in the string.
int	length()	Number of characters in the string.
int	compareTo(`String anotherString`)	Results in negative, zero or positive depending on the lexicographical ordering of the string and the argument anotherString using the rules of the default locale(explained below).
int	compareToIgnoreCase(`String str`)	Compares two strings lexicographically, ignoring case differences.
String	concat(`String str`)	Concatenates the specified string to the end of this string.
int	indexOf(`int ch`)	Returns the index within this string of the first occurrence of the specified character. There are three other variants of indexOf
String	replace(`char oldChar, char newChar`)	Returns a new string object resulting from replacing all occurrences of `oldChar` in this string with `newChar`. Uses the old string to return if no match found.
String	substring `(int beginIndex, int endIndex)`	Returns a new string object that is a substring of this string, from beginIndex to (endIndex-1)
String	toLowerCase()	Converts all of the characters in this `String` to lower case using the rules of the default locale.
String	toUpperCase()	Converts all of the characters in this `String` to upper case using the rules of the default locale.

In the above operations, the original string is untouched and a new object is created and returned.
In the above table `compareTo`, `toLowerCase, toUpperCase` uses the default "locale" for the operation. A Locale object represents a specific geographical, political, or cultural region. An operation that requires a Locale to perform its task is called *locale-sensitive* and uses the Locale to tailor information for the user. For example, displaying a number is a locale-sensitive operation--the number should be formatted according to the customs/conventions of the user's native country, region, or culture. In the above case, the notion of upper case letters and lower case letters are locale sensitive.

Input-Output

There is a notion of sequence while input and output in Java. The strings are output in the same sequence as given by the print statements. In Java, there are stream objects. The objects to be output, are put in to the output stream (can be considered like a buffer or pipe), and will be output in the same sequence. In a similar way, the input is put in to the input stream from the keyboard and the program can access the input stream to access the input data.

For input output we use the System class, (which was used in `System.out.print` for output). The input output functionalities is done by 3 static stream variables `in` (for input), `out` and `err` (both for output). So for the command:

`System.out.println`:

System is the class, out is the static stream object and println is the method.

Contrary to `System.out` for output, `System.in` is not very direct. The input stream considers everything as a stream of characters. So we have to scan through the input and extract the various data types like integer, float, string, char etc, from it. Scanner does precisely this function.

Scanner

A `Scanner` breaks its input into tokens using a delimiter pattern, which by default matches whitespace. The resulting tokens may then be converted into values of different types using the various `next` methods. It scans through the argument, which can be an input stream (`System.in`), a string or a file. In total `Scanner` has 8 constructors apart from the default and copy constructor. We have to import `the java.util` package to use the scanner.

For example

```
Scanner sc = new Scanner(System.in);     //scanner object creation
int i = sc.nextInt();     //scans the next integer from the input.
```

Return Type	**Method**	**Function**
String	next()	Finds and returns the next complete token from this scanner.
String	nextLine()	Advances this scanner past the current line and returns the input that was skipped. i.e., from the start position of the scanner till the newline
boolean	nextBoolean()	Scans and returns the next token of the input into a boolean value and returns that value. Ignores the case.
byte	nextByte()	Scans the next token of the input as a byte. The input should be small enough to fit in to a byte.

int	nextInt()	Scans and returns the next token of the input as an int.
short	nextShort()	Scans and returns the next token of the input as a short.
long	nextLong()	Scans and returns the next token of the input as a long.
double	nextDouble()	Scans and returns the next token of the input as a double.
float	nextFloat()	Scans and returns the next token of the input as a float.

Apart from these methods, there are many other methods for scanner. There is a variant of the above method which checks the next token to be of various types. For eg. `hasNextInt` returns true if the next token in this scanner's input can be interpreted as an int value in the default radix using the `nextInt`() method. Similar methods exists for other data types.